eBASH:
ECRYPT Benchmarking of All Submitted Hashes

http://bench.cr.yp.to/ebash.html

D. J. Bernstein
University of Illinois at Chicago

Joint work with:
Tanja Lange
Technische Universiteit Eindhoven

ECRYPT, VAMPIRE

European Union has funded
ECRYPT I network (2004–2008),
ECRYPT II network (2008–2012).

ECRYPT's "virtual labs" include
many universities, companies.

VAMPIRE is the
"Virtual Application and
Implementation Lab" led by
Tanja Lange (Eindhoven),
Christof Paar (Bochum).

I:

PT Benchmarking

Submitted Hashes

//bench.cr.yp.to

n.html

Bernstein

sity of Illinois at Chicago

work with:

Lange

sche Universiteit Eindhoven

ECRYPT, VAMPIRE

European Union has funded ECRYPT I network (2004–2008), ECRYPT II network (2008–2012).

ECRYPT's "virtual labs" include many universities, companies.

VAMPIRE is the "Virtual Application and Implementation Lab" led by Tanja Lange (Eindhoven), Christof Paar (Bochum).

eBATS

2006: V

("ECRY

of Asyn

measur

encrypt

2008: V

("ECRY

of All S

eBACS

of Cryp

include

http:/

marking
Hashes

cr.yp.to

ois at Chicago

ersiteit Eindhoven

ECRYPT, VAMPIRE

European Union has funded ECRYPT I network (2004–2008), ECRYPT II network (2008–2012).

ECRYPT's "virtual labs" include many universities, companies.

VAMPIRE is the "Virtual Application and Implementation Lab" led by Tanja Lange (Eindhoven), Christof Paar (Bochum).

eBATS, eBASH,

2006: VAMPIRE
("ECRYPT Benc
of Asymmetric S
measuring efficie
encryption, signa

2008: VAMPIRE
("ECRYPT Benc
of All Submitted

eBACS ("ECRYP
of Cryptographic
includes eBATS,

http://bench.

cago

ndhoven

ECRYPT, VAMPIRE

European Union has funded
ECRYPT I network (2004–2008),
ECRYPT II network (2008–2012).

ECRYPT's "virtual labs" include
many universities, companies.

VAMPIRE is the
"Virtual Application and
Implementation Lab" led by
Tanja Lange (Eindhoven),
Christof Paar (Bochum).

eBATS, eBASH, eBACS

2006: VAMPIRE started e
("ECRYPT Benchmarking
of Asymmetric Systems"),
measuring efficiency of pu
encryption, signatures, DH

2008: VAMPIRE started e
("ECRYPT Benchmarking
of All Submitted Hashes")

eBACS ("ECRYPT Bench
of Cryptographic Systems"
includes eBATS, eBASH,

http://bench.cr.yp.to

ECRYPT, VAMPIRE

European Union has funded ECRYPT I network (2004–2008), ECRYPT II network (2008–2012).

ECRYPT's "virtual labs" include many universities, companies.

VAMPIRE is the "Virtual Application and Implementation Lab" led by Tanja Lange (Eindhoven), Christof Paar (Bochum).

eBATS, eBASH, eBACS

2006: VAMPIRE started eBATS ("ECRYPT Benchmarking of Asymmetric Systems"), measuring efficiency of public-key encryption, signatures, DH.

2008: VAMPIRE started eBASH ("ECRYPT Benchmarking of All Submitted Hashes").

eBACS ("ECRYPT Benchmarking of Cryptographic Systems") includes eBATS, eBASH, more.

`http://bench.cr.yp.to`

PT, VAMPIRE

ean Union has funded
PT I network (2004–2008),
PT II network (2008–2012).

PT's "virtual labs" include
universities, companies.

IRE is the
al Application and
nentation Lab" led by
Lange (Eindhoven),
of Paar (Bochum).

eBATS, eBASH, eBACS

2006: VAMPIRE started eBATS ("ECRYPT Benchmarking of Asymmetric Systems"), measuring efficiency of public-key encryption, signatures, DH.

2008: VAMPIRE started eBASH ("ECRYPT Benchmarking of All Submitted Hashes").

eBACS ("ECRYPT Benchmarking of Cryptographic Systems") includes eBATS, eBASH, more.

`http://bench.cr.yp.to`

eBASH

eBASH
49 imp
28 has

http:/
/resul
already
measur
94 mac

Each i
recomp
with va
to iden
for imp

PIRE

has funded
rk (2004–2008),
ork (2008–2012).

ual labs" include
s, companies.

tion and
Lab" led by
ndhoven),
ochum).

eBATS, eBASH, eBACS

2006: VAMPIRE started eBATS ("ECRYPT Benchmarking of Asymmetric Systems"), measuring efficiency of public-key encryption, signatures, DH.

2008: VAMPIRE started eBASH ("ECRYPT Benchmarking of All Submitted Hashes").

eBACS ("ECRYPT Benchmarking of Cryptographic Systems") includes eBATS, eBASH, more.

http://bench.cr.yp.to

eBASH → public

eBASH has alrea
49 implementatio
28 hash function

http://bench.c
/results-hash
already shows
measurements on
94 machine-ABI

Each implementa
recompiled 1201
with various com
to identify best w
for implementatio

d
-2008),
–2012).

nclude
ies.

y

eBATS, eBASH, eBACS

2006: VAMPIRE started eBATS (“ECRYPT Benchmarking of Asymmetric Systems”), measuring efficiency of public-key encryption, signatures, DH.

2008: VAMPIRE started eBASH (“ECRYPT Benchmarking of All Submitted Hashes”).

eBACS (“ECRYPT Benchmarking of Cryptographic Systems”) includes eBATS, eBASH, more.

http://bench.cr.yp.to

eBASH → public

eBASH has already collect
49 implementations of
28 hash functions in 14 fa

http://bench.cr.yp.to
/results-hash.html
already shows
measurements on 68 mach
94 machine-ABI combinati

Each implementation is
recompiled 1201 times
with various compiler optio
to identify best working op
for implementation, machi

eBATS, eBASH, eBACS

2006: VAMPIRE started eBATS (“ECRYPT Benchmarking of Asymmetric Systems”), measuring efficiency of public-key encryption, signatures, DH.

2008: VAMPIRE started eBASH (“ECRYPT Benchmarking of All Submitted Hashes”).

eBACS (“ECRYPT Benchmarking of Cryptographic Systems”) includes eBATS, eBASH, more.

`http://bench.cr.yp.to`

eBASH → public

eBASH has already collected 49 implementations of 28 hash functions in 14 families.

`http://bench.cr.yp.to/results-hash.html` already shows measurements on 68 machines; 94 machine-ABI combinations.

Each implementation is recompiled 1201 times with various compiler options to identify best working option for implementation, machine.

, eBASH, eBACS

VAMPIRE started eBATS
YPT Benchmarking
mmetric Systems”),
ring efficiency of public-key
tion, signatures, DH.

VAMPIRE started eBASH
YPT Benchmarking
Submitted Hashes”).

s (“ECRYPT Benchmarking
ptographic Systems”)
es eBATS, eBASH, more.

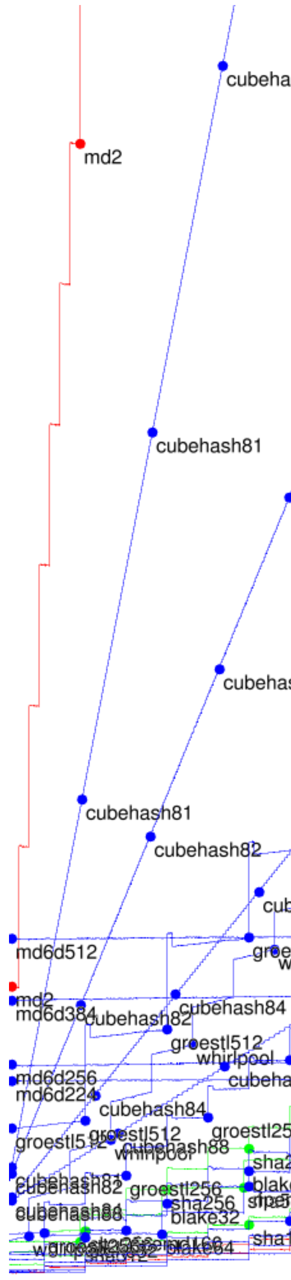
//bench.cr.yp.to

eBASH → public

eBASH has already collected 49 implementations of 28 hash functions in 14 families.

http://bench.cr.yp.to /results-hash.html already shows measurements on 68 machines; 94 machine-ABI combinations.

Each implementation is recompiled 1201 times with various compiler options to identify best working option for implementation, machine.



eBACS

. started eBATS
chmarking
ystems"),
ncy of public-key
tures, DH.

. started eBASH
chmarking
Hashes").

PT Benchmarking
Systems")
eBASH, more.

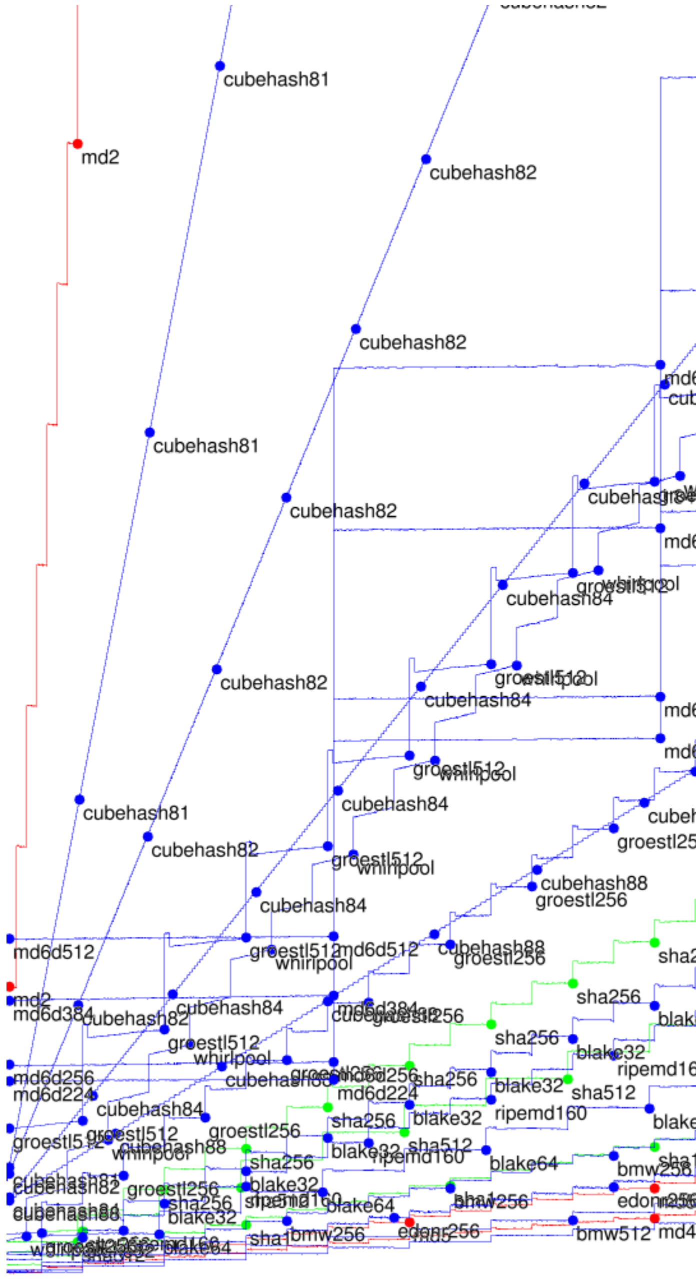
cr.yp.to

eBASH → public

eBASH has already collected 49 implementations of 28 hash functions in 14 families.

http://bench.cr.yp.to/results-hash.html already shows measurements on 68 machines; 94 machine-ABI combinations.

Each implementation is recompiled 1201 times with various compiler options to identify best working option for implementation, machine.



BATS

blic-key

.

BASH

.

marking

)

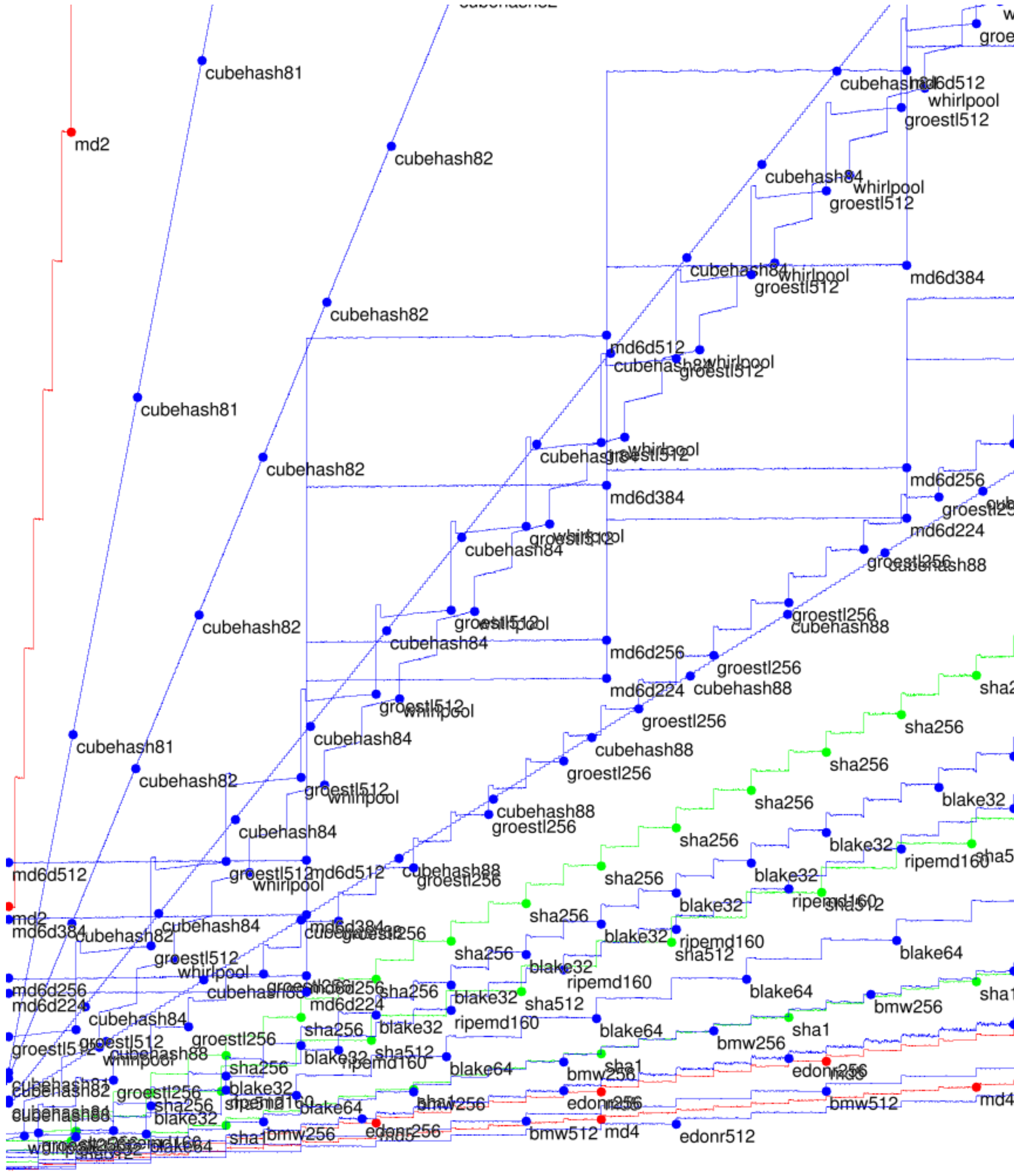
more.

eBASH → public

eBASH has already collected
49 implementations of
28 hash functions in 14 families.

http://bench.cr.yp.to
/results-hash.html
already shows
measurements on 68 machines;
94 machine-ABI combinations.

Each implementation is
recompiled 1201 times
with various compiler options
to identify best working option
for implementation, machine.

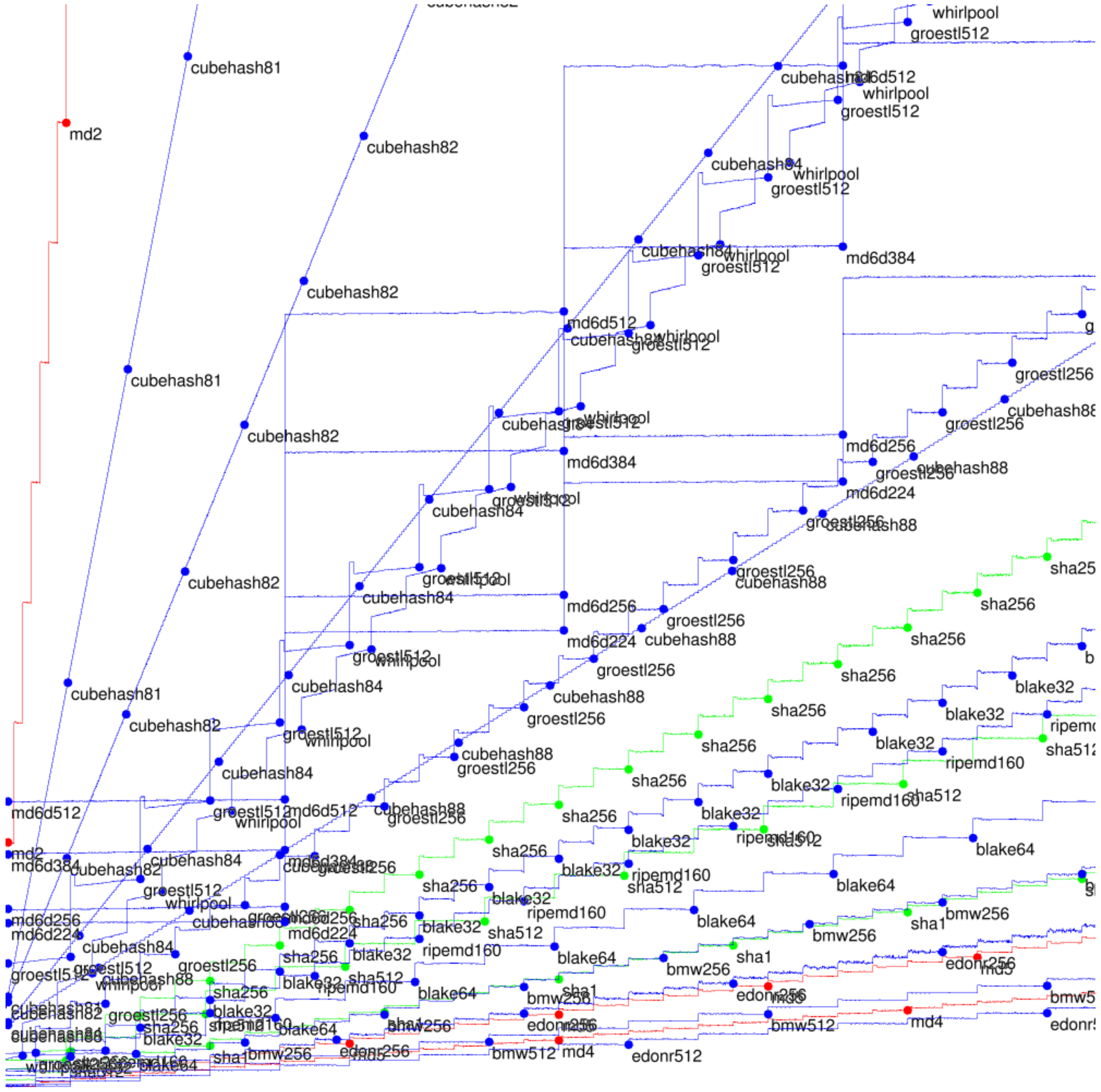


eBASH → public

eBASH has already collected
49 implementations of
28 hash functions in 14 families.

http://bench.cr.yp.to
/results-hash.html
already shows
measurements on 68 machines;
94 machine-ABI combinations.

Each implementation is
recompiled 1201 times
with various compiler options
to identify best working option
for implementation, machine.

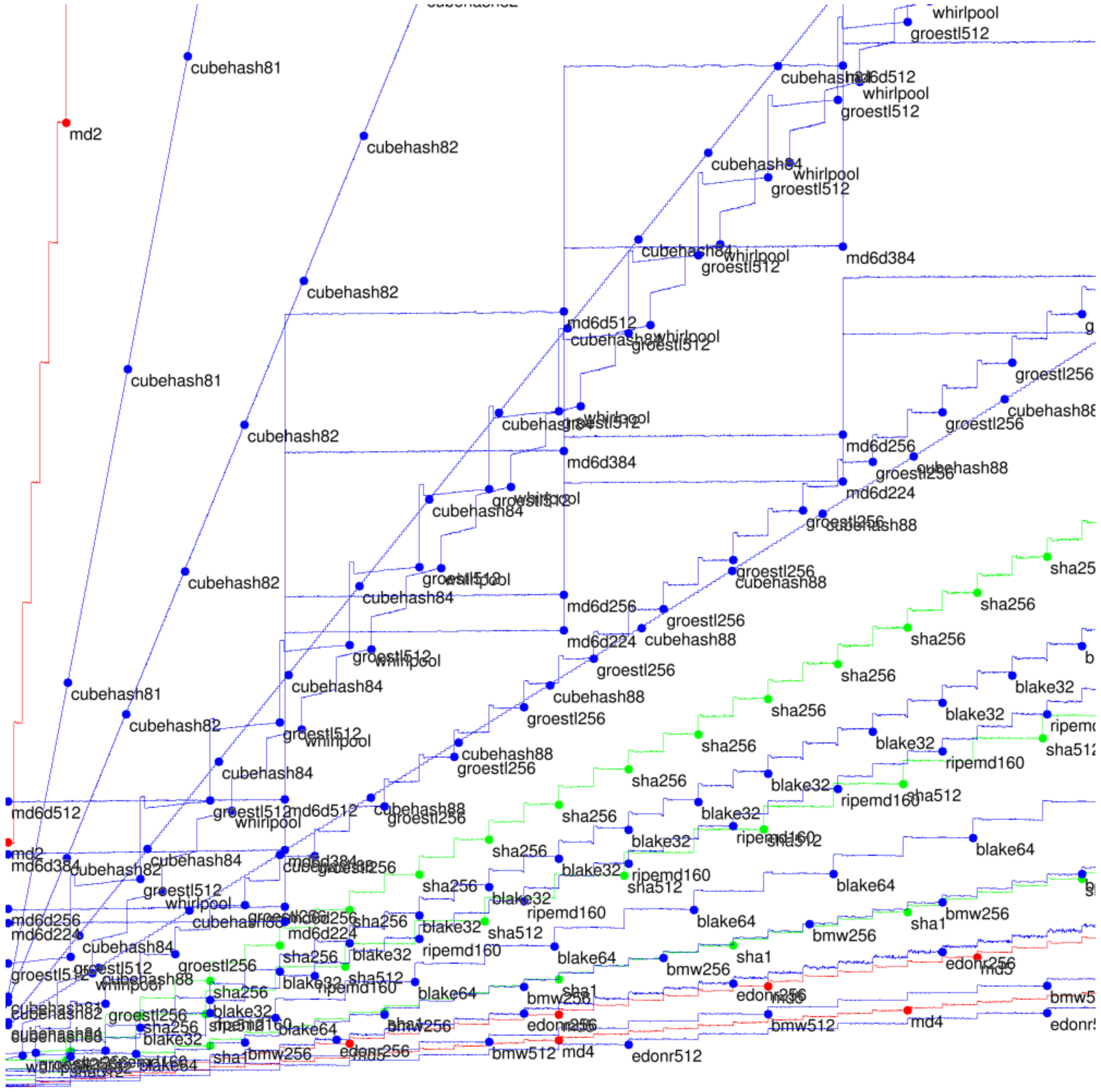


I → public

I has already collected
lementations of
h functions in 14 families.

//bench.cr.yp.to
lts-hash.html
y shows
rements on 68 machines;
chine-ABI combinations.

mplementation is
piled 1201 times
arious compiler options
tify best working option
plementation, machine.



Tables
of cycl
8-byte
64-byte
576-by
1536-b
4096-b
(extrap

Actuall
e.g. Re
e.g. Gr
0-byte
2-byte
4-byte
. . ., 20

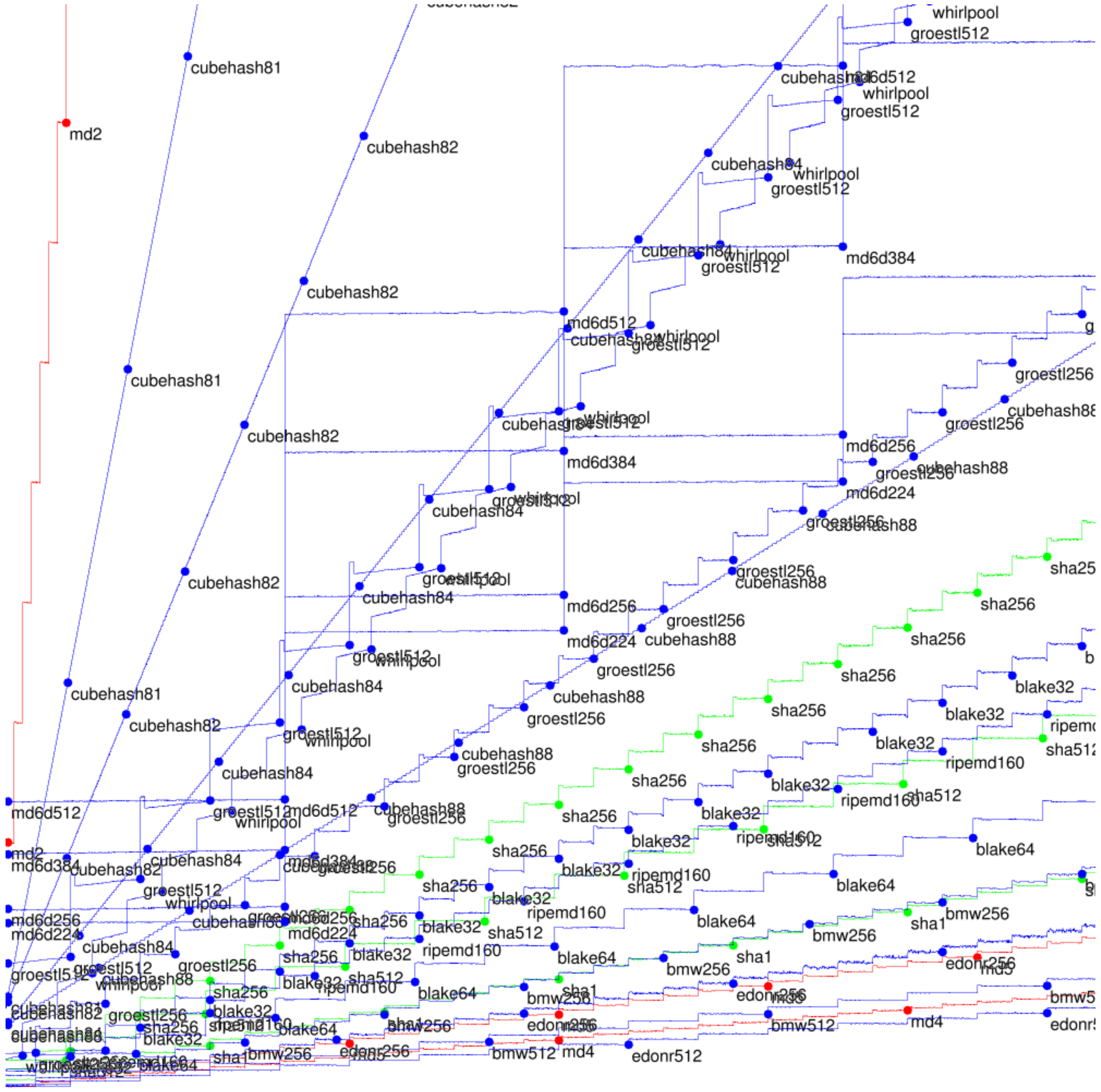
c

dy collected
ons of
s in 14 families.

`cr.yp.to`
`.html`

n 68 machines;
combinations.

ation is
times
piler options
working option
on, machine.



Tables show med
of cycles/byte to
8-byte message,
64-byte message,
576-byte messag
1536-byte messag
4096-byte messag
(extrapolated) lo

Actually have mu
e.g. Reports show
e.g. Graphs show
0-byte message,
2-byte message,
4-byte message,
. . ., 2048-byte m

ed

milies.

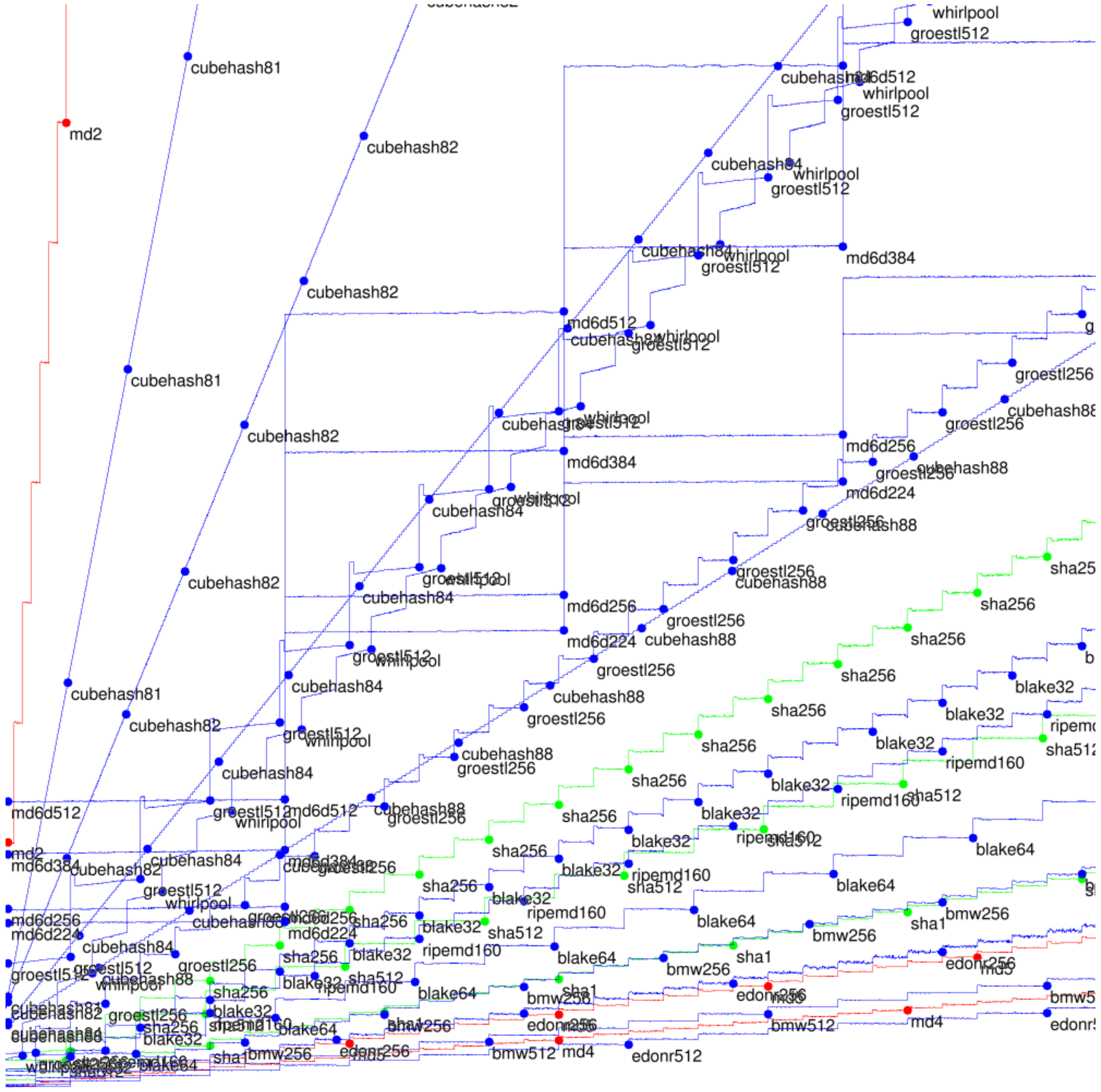
ines;

ons.

ons

tion

ne.



Tables show medians, quar

of cycles/byte to hash

8-byte message,

64-byte message,

576-byte message,

1536-byte message,

4096-byte message,

(extrapolated) long messag

Actually have much more c

e.g. Reports show best opt

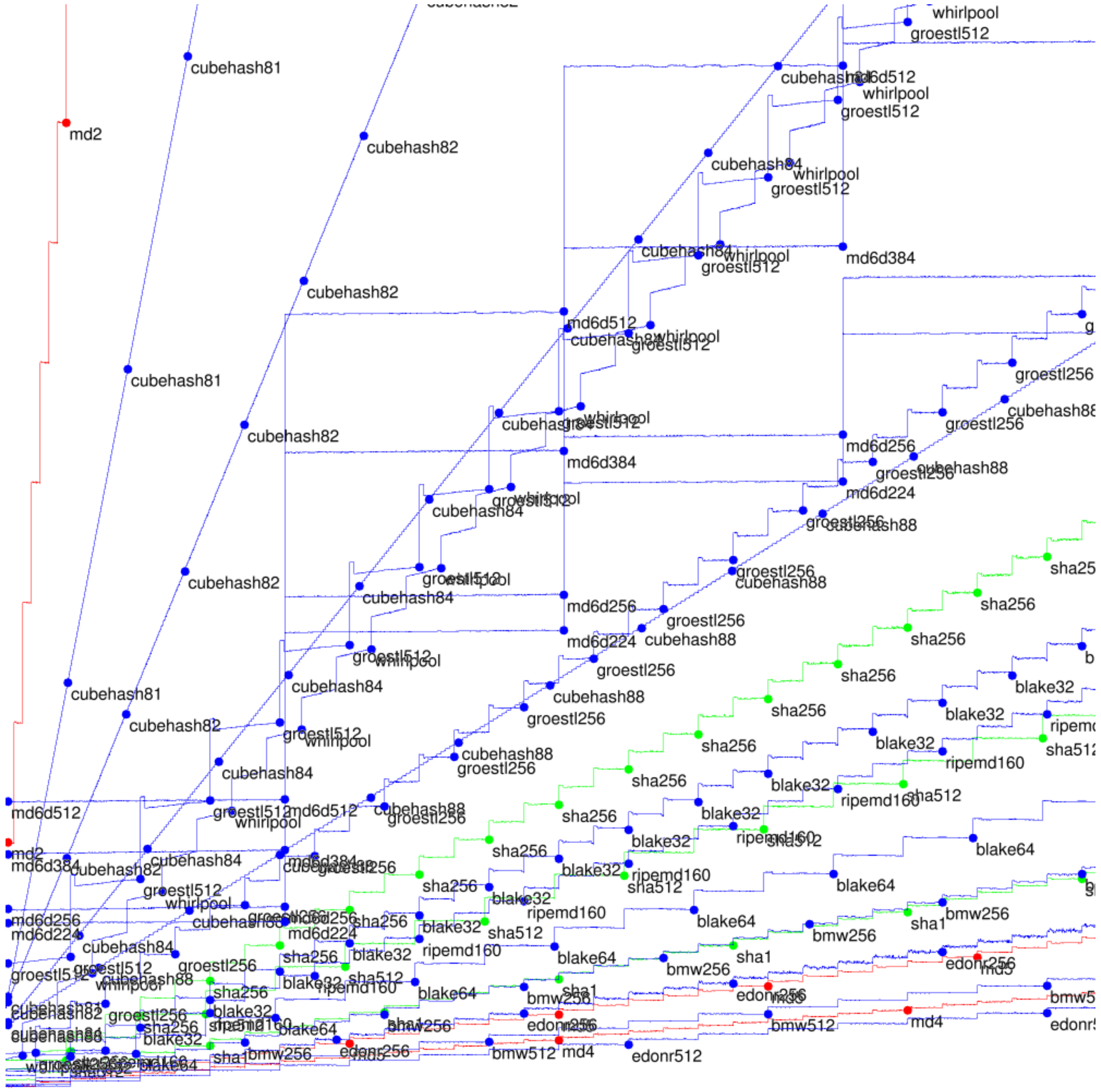
e.g. Graphs show medians

0-byte message, 1-byte me

2-byte message, 3-byte me

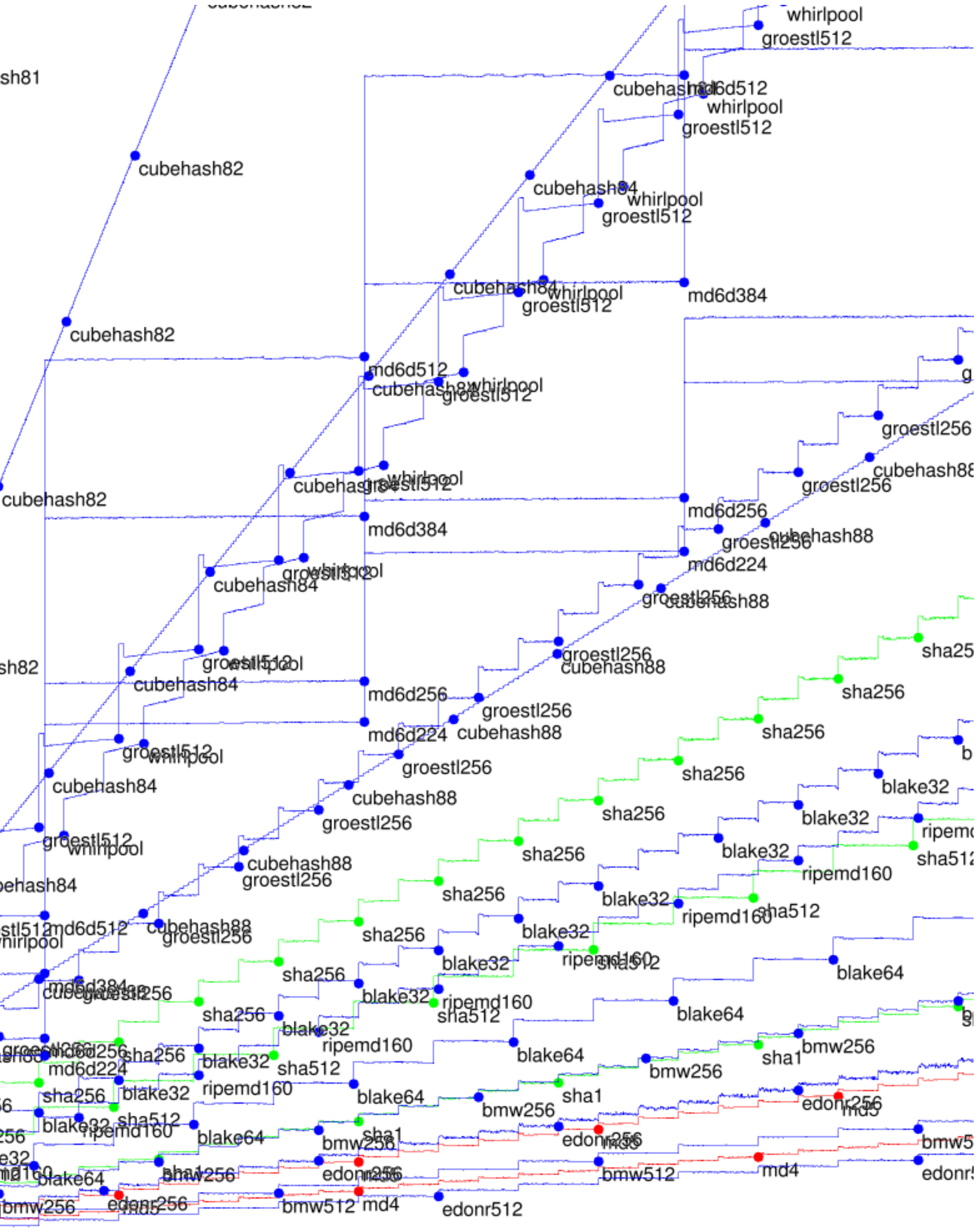
4-byte message, 5-byte me

. . ., 2048-byte message.



Tables show medians, quartiles of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . ., 2048-byte message.



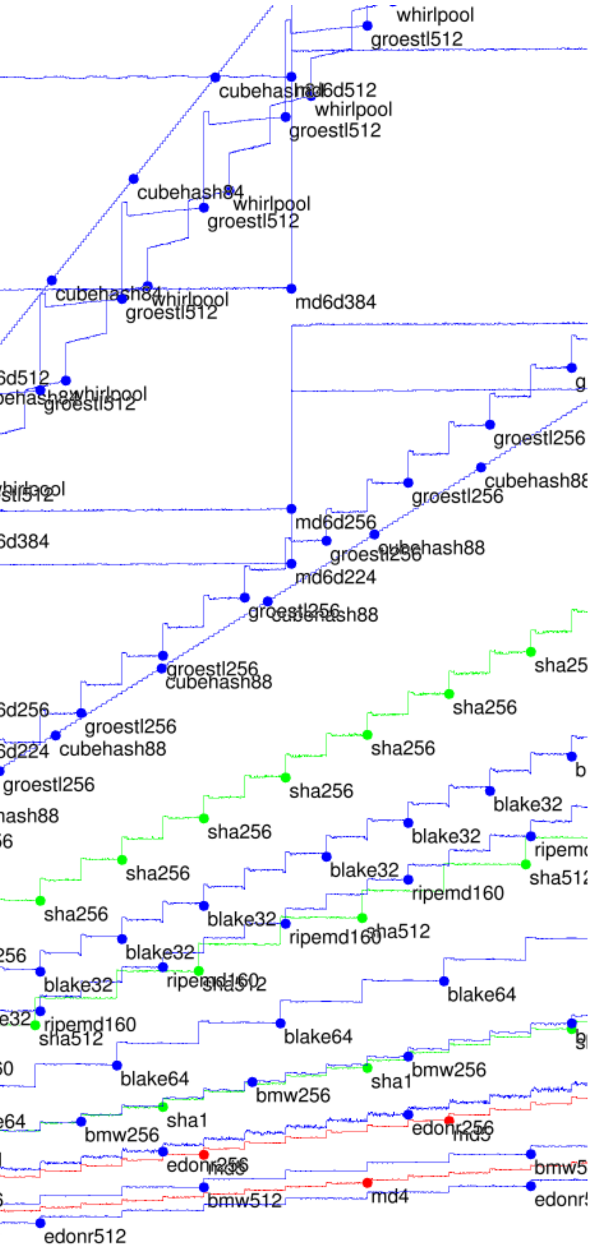
Tables show medians, quartiles of cycles/byte to hash 8-byte message, 64-byte message, 576-byte message, 1536-byte message, 4096-byte message, (extrapolated) long message.

Actually have much more data. e.g. Reports show best options. e.g. Graphs show medians for 0-byte message, 1-byte message, 2-byte message, 3-byte message, 4-byte message, 5-byte message, . . ., 2048-byte message.

e.g. 57

Core 2

25%
3.75
4.58
4.88
6.44
7.06
9.22
9.53
12.10
16.21
16.69
19.36
23.47
33.44



Tables show medians, quartiles of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . ., 2048-byte message.

e.g. 576 bytes, k
Core 2 Duo 6f6),

25%	50%	75
3.75	3.76	3.7
4.58	4.58	4.5
4.88	4.88	4.8
6.44	6.46	6.4
7.06	7.07	7.1
9.22	9.24	9.3
9.53	9.56	9.5
12.10	12.11	12.1
16.21	16.24	16.3
16.69	16.74	16.7
19.36	19.38	19.3
23.47	23.49	23.5
33.44	33.44	33.5

Tables show medians, quartiles of cycles/byte to hash 8-byte message, 64-byte message, 576-byte message, 1536-byte message, 4096-byte message, (extrapolated) long message.

Actually have much more data. e.g. Reports show best options. e.g. Graphs show medians for 0-byte message, 1-byte message, 2-byte message, 3-byte message, 4-byte message, 5-byte message, . . ., 2048-byte message.

e.g. 576 bytes, `katana` (21
Core 2 Duo 6f6), 64-bit AE

25%	50%	75%	hash
3.75	3.76	3.79	edonr5
4.58	4.58	4.58	md4
4.88	4.88	4.88	bmw51
6.44	6.46	6.46	md5
7.06	7.07	7.15	edonr2
9.22	9.24	9.31	bmw25
9.53	9.56	9.57	sha1
12.10	12.11	12.12	blake64
16.21	16.24	16.35	sha512
16.69	16.74	16.78	ripemd
19.36	19.38	19.38	blake32
23.47	23.49	23.53	sha256
33.44	33.44	33.51	groestl
			etc.

Tables show medians, quartiles of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . ., 2048-byte message.

e.g. 576 bytes, `katana` (2137MHz Core 2 Duo 6f6), 64-bit ABI:

25%	50%	75%	hash
3.75	3.76	3.79	edonr512
4.58	4.58	4.58	md4
4.88	4.88	4.88	bmw512
6.44	6.46	6.46	md5
7.06	7.07	7.15	edonr256
9.22	9.24	9.31	bmw256
9.53	9.56	9.57	sha1
12.10	12.11	12.12	blake64
16.21	16.24	16.35	sha512
16.69	16.74	16.78	ripemd160
19.36	19.38	19.38	blake32
23.47	23.49	23.53	sha256
33.44	33.44	33.51	groestl256
			etc.

show medians, quartiles
es/byte to hash
message,
e message,
te message,
yte message,
yte message,
polated) long message.

ly have much more data.
eports show best options.
aphs show medians for
message, 1-byte message,
message, 3-byte message,
message, 5-byte message,
48-byte message.

e.g. 576 bytes, `katana` (2137MHz Core 2 Duo 6f6), 64-bit ABI:

25%	50%	75%	hash
3.75	3.76	3.79	edonr512
4.58	4.58	4.58	md4
4.88	4.88	4.88	bmw512
6.44	6.46	6.46	md5
7.06	7.07	7.15	edonr256
9.22	9.24	9.31	bmw256
9.53	9.56	9.57	sha1
12.10	12.11	12.12	blake64
16.21	16.24	16.35	sha512
16.69	16.74	16.78	ripemd160
19.36	19.38	19.38	blake32
23.47	23.49	23.53	sha256
33.44	33.44	33.51	groestl256
			etc.

Submit

Define

```
#defi
```

dians, quartiles
hash

,
e,
ge,
ge,
ng message.

uch more data.
v best options.
v medians for
1-byte message,
3-byte message,
5-byte message,
essage.

e.g. 576 bytes, `katana` (2137MHz Core 2 Duo 6f6), 64-bit ABI:

25%	50%	75%	hash
3.75	3.76	3.79	edonr512
4.58	4.58	4.58	md4
4.88	4.88	4.88	bmw512
6.44	6.46	6.46	md5
7.06	7.07	7.15	edonr256
9.22	9.24	9.31	bmw256
9.53	9.56	9.57	sha1
12.10	12.11	12.12	blake64
16.21	16.24	16.35	sha512
16.69	16.74	16.78	ripemd160
19.36	19.38	19.38	blake32
23.47	23.49	23.53	sha256
33.44	33.44	33.51	groestl256
			etc.

Submitter → eB

Define output siz

```
#define CRYPT
```

rtiles

ge.

data.

ions.

for

ssage,

ssage,

ssage,

e.g. 576 bytes, `katana` (2137MHz Core 2 Duo 6f6), 64-bit ABI:

25%	50%	75%	hash
3.75	3.76	3.79	edonr512
4.58	4.58	4.58	md4
4.88	4.88	4.88	bmw512
6.44	6.46	6.46	md5
7.06	7.07	7.15	edonr256
9.22	9.24	9.31	bmw256
9.53	9.56	9.57	sha1
12.10	12.11	12.12	blake64
16.21	16.24	16.35	sha512
16.69	16.74	16.78	ripemd160
19.36	19.38	19.38	blake32
23.47	23.49	23.53	sha256
33.44	33.44	33.51	groestl256
			etc.

Submitter → eBASH

Define output size in `api.`

```
#define CRYPTO_BYTES
```

e.g. 576 bytes, `katana` (2137MHz Core 2 Duo 6f6), 64-bit ABI:

25%	50%	75%	hash
3.75	3.76	3.79	edonr512
4.58	4.58	4.58	md4
4.88	4.88	4.88	bmw512
6.44	6.46	6.46	md5
7.06	7.07	7.15	edonr256
9.22	9.24	9.31	bmw256
9.53	9.56	9.57	sha1
12.10	12.11	12.12	blake64
16.21	16.24	16.35	sha512
16.69	16.74	16.78	ripemd160
19.36	19.38	19.38	blake32
23.47	23.49	23.53	sha256
33.44	33.44	33.51	groestl256
			etc.

Submitter → eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```

e.g. 576 bytes, `katana` (2137MHz Core 2 Duo 6f6), 64-bit ABI:

25%	50%	75%	hash
3.75	3.76	3.79	edonr512
4.58	4.58	4.58	md4
4.88	4.88	4.88	bmw512
6.44	6.46	6.46	md5
7.06	7.07	7.15	edonr256
9.22	9.24	9.31	bmw256
9.53	9.56	9.57	sha1
12.10	12.11	12.12	blake64
16.21	16.24	16.35	sha512
16.69	16.74	16.78	ripemd160
19.36	19.38	19.38	blake32
23.47	23.49	23.53	sha256
33.44	33.44	33.51	groestl256
			etc.

Submitter → eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```

Define hash function in `hash.c`, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
       ,in,inlen*8,out);
  return 0; }
```

6 bytes, katana (2137MHz
Duo 6f6), 64-bit ABI:

50%	75%	hash
3.76	3.79	edonr512
4.58	4.58	md4
4.88	4.88	bmw512
6.46	6.46	md5
7.07	7.15	edonr256
9.24	9.31	bmw256
9.56	9.57	sha1
12.11	12.12	blake64
16.24	16.35	sha512
16.74	16.78	ripemd160
19.38	19.38	blake32
23.49	23.53	sha256
33.44	33.51	groestl256
		etc.

Submitter → eBASH

Define output size in api.h:

```
#define CRYPTO_BYTES 64
```

Define hash function in hash.c, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
    unsigned char *out,
    const unsigned char *in,
    unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
          ,in,inlen*8,out);
    return 0; }
```

Send t
the UR
with o
crypt
contai

Measu

Much
to do

More
http:/
/call-

atana (2137MHz

. 64-bit ABI:

%	hash
79	edonr512
58	md4
38	bmw512
46	md5
15	edonr256
31	bmw256
57	sha1
12	blake64
35	sha512
78	ripemd160
38	blake32
53	sha256
51	groestl256
	etc.

Submitter → eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```

Define hash function in `hash.c`, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
       ,in,inlen*8,out);
  return 0; }
```

Send to the mail

the URL of a ta

with one director

`crypto_hash/yo`

containing `hash.`

Measurements m

Much easier thar

to do your own b

More details and

`http://bench.c`

`/call-hash.htm`

.37MHz
3I:

12

2

56
6

4

160
2

256

Submitter → eBASH

Define output size in api.h:

```
#define CRYPTO_BYTES 64
```

Define hash function in hash.c, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
       ,in,inlen*8,out);
  return 0; }
```

Send to the mailing list the URL of a tar.gz with one directory crypto_hash/yourhash/ containing hash.c etc.

Measurements magically ap

Much easier than trying to do your own benchmark

More details and options:
http://bench.cr.yp.to
/call-hash.html

Submitter → eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```

Define hash function in `hash.c`, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
       ,in,inlen*8,out);
  return 0; }
```

Send to the mailing list
the URL of a `tar.gz`
with one directory
`crypto_hash/yourhash/ref`
containing `hash.c` etc.

Measurements magically appear!
Much easier than trying
to do your own benchmarks.

More details and options:
`http://bench.cr.yp.to`
`/call-hash.html`